Communication-Aware Prediction-Based Online Scheduling in High-Performance Real-Time Embedded Systems

Baptiste Goupille-Lescar, Eric Lenormand, Christine Morin, Nikos Parlavantzas

Communication-Aware Prediction-Based Online Scheduling in High-Performance Real-Time Embedded Systems

Baptiste Goupille–Lescar[1,2], Eric Lenormand[1], Nikos Parlavantzas[2,3], and Christine Morin[2]

[1] Thales Research & Technology, 1 av. Augustin Fresnel, 91120 Palaiseau, France
{baptiste.goupillelescar, eric.lenormand}@thalesgroup.com
https://www.thalesgroup.com/en
[2] Inria, IRISA, 263 av. General Leclerc, 35042 Rennes, France
christine.morin@inria.fr
https://www.inria.fr/en/
[3] INSA Rennes, IRISA, 263 av. General Leclerc, 35042 Rennes, France
nikos.parlavantzas@irisa.fr

Abstract. Current high-end, data-intensive real-time embedded sensor applications (e.g., radar, optronics) require very specific computing platforms. The nature of such applications and the environment in which they are deployed impose numerous constraints, including real-time constraints, and computing throughput and latency needs. Static application placement is traditionally used to deal with these constraints. However, this approach fails to provide adaptation capabilities in an environment in constant evolution. Through the study of an industrial radar use-case, our work aims at mitigating the aforementioned limitations by proposing a low-latency online resource manager derived from techniques used in large-scale systems, such as cloud and grid environments. The resource manager introduced in this paper is able to dynamically allocate resources to fulfill requests coming from several sensors, making the most of the computing platform while providing guaranties on non-functional properties and Quality of Service (QoS) levels. Thanks to the load prediction implemented in the manager, we are able to achieve a 83% load increase before overloading the platform while managing to reduce ten times the incurred QoS penalty. Further methods to reduce the impact of the overload are as well as possible future improvements are proposed and discussed.

Keywords: Embedded systems, Real-time, Scheduling, Dynamic resource management

1 Introduction

Nowadays, with the increasing demand for high-performance computing and smart sensing, high-end embedded system designers are facing an increasing number of challenges. Indeed, the targeted platforms must respect a great number of non-functional constraints, such as Size, Weight and Power (SWaP), real-time computing and cost constraints. Currently, most embedded systems meet these constraints by the use of dedicated components and static resource allocation approaches based on worst-case scenarios. This method, coupled with the emergence of workloads integrating hard, soft real-time and best-effort applications and the increase of their variability, results in massive over-provisioning and under-utilization of resources. Moreover, while this method allows the design of efficient and reliable systems, it nearly eliminates their adaptation and evolution capabilities by preventing the deployment of highly variable or opportunistic applications for smart sensing. To

address these limitations, this paper proposes a smart resource management system, able to fulfill low-latency run-time requests for application execution while providing non-functional guarantees. Timing properties are considered in this paper, while other properties such as heat dissipation, will be addressed in future work. To achieve high performance gains while guaranteeing timing properties, our contribution is inspired by large-scale resource managers found in cloud or grid infrastructures, making the most of application profiling to enable high-level predictability, low mapping latency as well as high resource utilization. This work is supported by several industrial use-cases, including an Active Electronically Scanned Array (AESA) radar use-case, which is detailed in this paper. To fit the targeted context, a complete simulation framework has been implemented. The results obtained through simulations show that our mapping method results in improvements in both performance and predictability of the system.

This paper is organized as follows: Section 2 provides a quick description of the motivating use-case context. In Section 3, related research from both the embedded and large-scale systems communities are presented. Section 4 describes the models and methods used in our approach. In Section 5 the simulation framework created is explained and simulation results are analyzed. Section 6 introduces future work and discusses several open questions. Finally, Section 7 draws some conclusions and perspectives.

2 Use-Case Specific Context

Our use-case deals with optimizing the computing resource utilization of a multi-function electronically-steered surface radar and has been characterized with the support of domain experts. This equipment can typically be installed on the ground vehicles or on some surface ships such as frigates, with the mission of detecting, tracking and identifying objects of interest. This is done by illuminating narrow angular sectors by one or more sequences of known signals (waveforms) and processing the returned echoes captured on a large number of receivers. Such an observation is commonly referred as "dwell". Dwells can be of several types, depending on their objective (e.g., scanning an unknown sector, tracking a known target) and make use of different waveforms, each with its known duration. The radar antenna is constantly kept busy and scans the whole radar angular range by successively sending signals with the appropriate waveforms. The processing of received echoes is traditionally split into two distinct phases:

- A front-end one that applies fixed filtering functions on the digitized received channels. It processes a massive bandwidth of input data, and selects a limited set of points of interest by using appropriate detection thresholds. This phase is highly computation-intensive and computing times are deterministic.
- A back-end phase whose goal is to extract operational information (e.g., position, speed, trajectory, nature of target) from the detection results. Among functions executed in the back-end are the extraction, which delivers so-called "plots" that characterize one or more targets in precise directions of space, and tracking, which consolidates plots issued by extraction and builds trajectories of targets.

Considering the regularity of the front-end workload, a near-optimal mapping (i.e., resource allocation) can be found at design-time and used during run-time processing. Thus, front-end tasks are not targeted by this study as their high predictability limits the potential performance gain from the integration of dynamic elements. On the contrary, static design-time mapping methods loose efficiency when applied to functions with data-dependent complexities, which is the case for

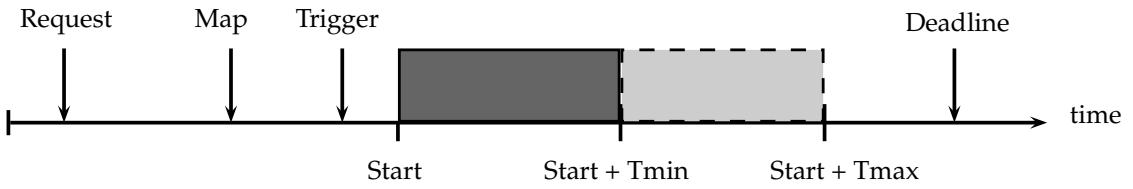

Fig. 1: Application execution timeline

most back-end applications. Functions like extraction have a largely variable computational cost, depending on the type of dwell as well as the number and configurations of targets that must be discriminated from the front-end detection. The actual run-time of each extraction is only known at the end of its execution. However, this cost is bounded by known minimum/maximum values related to each dwell type.

The computing platform used for back-end processing must fit within a specific SWaP budget. In our case, the targeted computing platform consists of 4 racks each one hosting several boards with several processing elements and a shared memory. All these elements are linked together by an Ethernet network using a star topology with one central router connected with the routers present in each rack. In addition to SWaP constraints, processing latencies matter, as observing and tracking targets needs to be kept in pace with their kinematics. The combination of variable computation times (and data transfer), constrained platform and real-time computing renders design-time mapping policies inefficient and drives the need for agile methods. Another goal of the introduction of online resource management in traditionally static systems is to serve as an enabler for the implementation of new applications with varying resource demand or unknown completion time.

To add adaptation capabilities to the system, we experiment with a resource management functionality that receives sporadic sequences of requests, each corresponding to a dwell that will be input to one of the 4 radar antennas. The *trigger* (see Fig.1) represents the moment at which the data will be available for the back-end (e.g. extraction) applications, after waveform emission/reception and front-end processing. The resource manager allocates the different back-end computation and communication activities of the dwell application onto the platform resources so that they can be executed before their deadline. For reasons discussed later, this placement decision *Map* is taken just before the trigger reception. After having been placed, an application's execution effectively starts only when all its dependencies are met and the targeted processing elements are free, which ideally happens at $T = trigger$. It then executes for a time depending on the data content before sending the results and freeing the occupied computing and communication resources. Furthermore, note that the processing of separate dwells (addressing different regions in space) can be done independently from each other.

Their execution is mandatory, every request must be fulfilled, and is non-preemptive. As of now, the resource management system is only seen as an executant and does not have the opportunity to reject or terminate an application. Once a request has been received, the associated application is executed in its entirety.

3 Related Work

As we are addressing high-performance computing problems in an embedded environment, the following related work is divided into two parts: studies targeting embedded systems with SWaP and timing constraints and studies addressing the placement of similar application models on large-scale systems.

3.1 Embedded Systems

We first take a look at works that consider similar computing platforms with timing constraints. In our study, as in numerous embedded systems design problems, the real-time execution of applications is a major concern. A great number of published works aim at providing timing guaranties for hard real-time applications. Works aiming at aeronautic or automotive certification reject dynamic methods and focus on single-processor architectures, as in [15] or [19], to maximize the system's predictability. In recent years, there has been a growing interest in both multi-processor architectures and mixed-criticality workloads. This led to numerous publications [3] and several European research projects [1] [2]. However these studies, such as [11] or [22], only address design-time schedulability analysis.

In [21], run-time adaptation consists in mapping long-lasting jobs onto a multi-processor system-on-chip architecture by making use of configurations elaborated at design-time. However, in our case the run-time management system operates in a non-periodic context by taking very frequent mapping decisions for computing jobs with a very limited duration. Similarly, the authors of [9] use design-time exploration results to dynamically adapt the application run-time setting depending on the context. While both these works show promising results, our workload's temporal behavior makes it impossible to adapt the resource allocated to an application once it started. The fact that all applications of our use-case are short-lived and must meet their deadline makes it impossible to modify the set of resources they are deployed on after the applications have started their execution.

3.2 Large-Scale Systems

While studies in the embedded community essentially focus on real-time control workloads or signal/video processing applications, our use-case's workload is much closer to workflows encountered in large-scale systems, such as cloud computing. With the democratization of cloud computing and an increasing need for efficient scientific simulations, more and more works address the scheduling of workflows in HPC systems [16]. These workflows are often materialized as Directed Acyclic Graphs (DAGs) for which computing and communication resources are allocated until their completion. Among the numerous publications, two main categories of resource management systems can be identified:

- Purely dynamic approaches with close to no prior knowledge of the applications and relying on cloud elasticity to compensate for QoS run-time variation by allocating/releasing and/or migrating Virtual Machines (VMs) [27] [7].
- Static methods relying on exhaustive or evolutionary algorithms to find an optimal placement before executing the application. Commonly used methods include genetic, ant colony and Min-Min algorithm variations [25] [23] [14] [6] [4].

While both of these approaches have merits and can achieve great results in cloud or grid computing, none are directly applicable to our use-case for several reasons. First, due to the arrival rate of requests, a placement solution must be found in a few milliseconds, which prevents us from using, for example, genetic or particle swarm algorithms. Furthermore, while some methods using online model derivation, such as [8] can show great results, they are only applicable to long-lasting applications. Then, the use-case's applications being non-preemptive and the resources not being virtual makes it impossible to rely on migration or virtual machine re-dimension mechanisms.

A fair number of studies try to incorporate predictability via the use of priorities or isolation mechanisms, as in [13][17]. Unfortunately, due to the ever-changing nature of cloud environments, most "real-time" cloud resource managers react to deadline misses and do not prevent them. Thus, while no management system is actually able to provide hard real-time guaranties, there exist few satisfying solutions able to accurately provide soft real-time guaranties when targeting data streaming. However, while it is manageable to obtain these timing properties *inside* a cloud environment, it is noticeably more difficult to do so when considering communications with users as data have to go through heavy software stacks as well as non-deterministic communication protocols [10]. In [5], while the authors use similar workload and resource reservation mechanisms to ours, they only target bag of tasks applications, for which they have control over the parallelism level, do not consider SWaP constraints and can potentially reserve an infinite number of VMs.

Finally, while the problem has already been addressed for single processor systems [18], only few works actually target non-preemptive resource allocation for stochastic workloads on multi-processor platforms [24] [12]. Moreover, they are purely mathematical approaches and totally abstract the computing platform by considering a set of identical unrelated machines.

To conclude, while a significant number of works seem to address a similar problem, the specificity of the considered use-case applications and the computing platform makes it impossible to simply adapt existing solutions.

4 Our Approach

In this section we first describe the overall functioning of our resource manager, consisting of the *mapper* and *predictor* components. Afterwards, this section describes the proposed mapping heuristic. Our final objective being to maximize the QoS provided to the radar's operator, a QoS-aware extension of our method is finally introduced.

4.1 Mapping Process

The overall mapping process is shown Fig.2 which serves as a guideline for this section. As explained in Section 2, the resource manager receives requests from the radar every few milliseconds (*1* in Fig.2). Each of these requests contains the type of application to execute, its identifier, its priority, the input and output memories and the time at which the data will be available in the input memory. The application's type defines its structure (i.e., directed acyclic graph) and profiling data, such as computing time and input and output data volume for which only minimal and maximal values are known. Note that priorities are only partially correlated to the application's type because they are dynamically attributed depending on the context (e.g., threat level of a target).

It is necessary to find a valid mapping solution for these requests before the data is available to reduce latency as well as to avoid congestion in the input memory. Furthermore, only probabilistic execution times of each computation and communication activity are known when evaluating

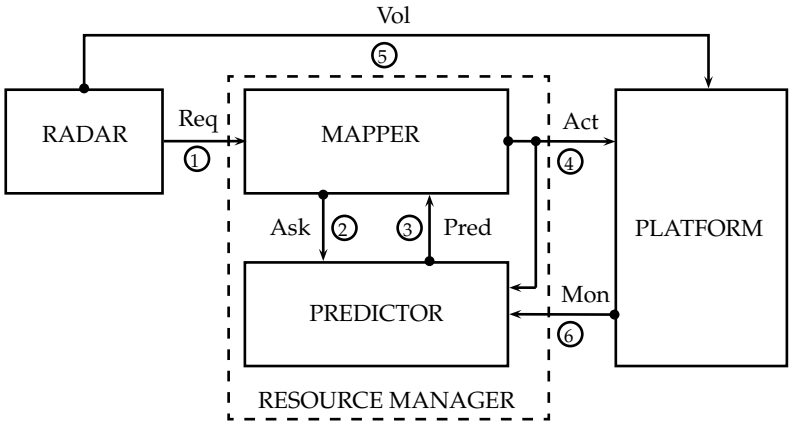

Fig. 2: Simplified model view

placement possibilities. Thus, unlike numerous approaches present in the literature, it is necessary to take a decision based not on the current state of resources but on an estimation of their future availability. To this extend, the *Mapper* interacts with the load *Predictor* (*2*)(*3*) to find a suitable placement for both communication and computation activities of the application (i.e., nodes of the DAG). Once a decision has been made, the application's activities are sent to their respective hosts (*4*). They are executed on the platform using data from the radar (*5*) to determine their actual execution timings. Finally, monitoring signals from the platform (*6*) are then exploited to update the load prediction model.

4.2 Prediction Model

Computing time predictions are handled by our *Predictor* module. Its role consists in keeping an up-to-date simplified overview of the platform's processors, links and memories state as well as queued activities to provide the *Mapper* with execution time predictions. Keeping in mind that a valid mapping solution must be found in a few milliseconds, the simulation of the actual computing platform in its entirety is far too computationally intensive to consider in detail while looking for a run-time solution. Indeed, since the platform comprises multiple shared resources (memories, communication links), estimating the impact of the deployment of an application on all the applications already running (or queued) represents a significant amount of computation, impossible to terminate in a few milliseconds on an embedded platform. Thus, the *Predictor* maintains a simplified model of the real architecture containing every resource (processors, memories, links) as well as the activities queued for execution in each of these resources. The execution time predictions are made without taking into account some of the resource access conflicts happening in the actual platform. The architecture model inside the predictor is kept up-to-date using both the *Mapper*'s placement decisions in addition to the platform's monitoring signals. The interactions between *Mapper*, *Platform* and *Predictor* can be seen as a MAPE (Monitoring, Analysis, Planning and Execution) control loop often found in large-scale systems management systems. Some adjustments are necessary to this loop as, while it is possible to take a mapping decision at the time a request is received, it may

be profitable to wait for a more precise prediction since the application's first activity will only be executed tens or hundreds of milliseconds later, when the data is available. The earlier a decision is taken, the less accurate the execution time prediction is as we would be making predictions on top of predictions since execution timings of an application depend on estimations of other applications' end times. On the contrary, taking a decision at the last minute means a reduced time to evaluate possible solutions. Thus, a trade-off must be found between the number of possible mappings we can evaluate versus the accuracy of these evaluations to find the best possible solution.

It is important to note that it is possible to make some assumptions on the system's evolution thanks to its quite regular request submission pattern; it would be impossible to anticipate its behavior if it was completely random. The system receives aperiodic sequences of mapping requests with an average interval between them in the range of a few milliseconds. Indeed, the rate at which the mapper receives requests is correlated with the radar antennas emission rate and, in practical use-cases, the antennas operate close to their maximum capabilities at all times. As mentioned before, their execution times can greatly vary depending on the data to process which can not be known before execution. In summary, we are faced with a system whose dynamic behavior enables short term predictions ranging from a few milliseconds to tens of milliseconds.

4.3 Proposed Mapping Heuristics

The considered application model is as follows: Each application is seen as a directed acyclic graph $APP = (A, D, prio, trigger, dl, input, output)$ having a priority $prio$, a memory $input$ at which necessary data will be available at $t = trigger$ and the memory $output$ in which the results have to be stored before the deadline dl. A represents a set of computing activities linked by execution order constraints, or dependencies D. An Activity $A_n = (vol_{comp_{max}}, vol_{comp_{min}}, vol_{in_{max}}, vol_{in_{min}}, vol_{out_{max}}, vol_{out_{min}}, \alpha)$ has minimum/maximum values of the number of instructions to execute (vol_{comp}), the data input vol_{in} and output vol_{out} volumes as well as a memory access ratio α. The memory access ratio α of a computing node determines the minimum memory bandwidth needed by a target processor to perform the computation of this activity at maximum speed. For an application to be schedulable, we assume that $\forall APP, T_{max} < (deadline - trigger)$, T_{max} being the sum of maximum computation times of the activities part of the critical path of APP. In other words, applications have *slack*, meaning most of them can be slightly delayed and still respect their timing constraints.

Concerning the architecture, reasoning is made on the architecture model maintained by the predictor and defined as follows. The platform $PF = (P, M, L)$ is seen as a set of processors P, memories M and links L. Each processor P_n possess a computing capacity expressed in instructions per second in addition to its type, each memory M_n a maximum bandwidth and each link L_n a maximum throughput expressed in Bytes per second. At run-time the architecture runs several applications. Processors (resp. communication links) can only serve one computation (resp. communication) at a time. Memories on the contrary potentially share their bandwidth between different computations and communications, which may result in slowing down some of them in case of excessive demands.

Static mapping This mapping strategy is used as a reference for comparison with dynamic strategies shown below. In this context, processors are allocated to successive requests in each rack on a systematic round-robin basis, without considering the computing cost or priority of each requested activity.

Prediction-Based Load Balancing This dynamic strategy consists in allocating at run-time the best communication path and computing elements in the machine, by taking account of an estimated load status of the different resources of the architecture. Its objective is to minimize the end time of the next request to be executed. This is done by investigating all sets of processors $SP_n \subset P$ able to host the different computation activities to execute. For each of them, since an application request carries information on input and output memories, it is possible to identify and generate an adequate set of communication activities to transfer data from the input memory to computing activities hosts' memories and, finally, to the output memory. From this set of activity sets, the mapper then estimates for each of them the worst-case completion times of the application request according to the architecture resources used and their estimated current workload. The predictor is used there, providing for each resource an estimation of the completion time of its last queued activity, or the indication that it has no activity still running. This is then used to evaluate the completion time of the new candidate activity request, and elect the hosting processor and associated communication path that achieves the earliest predicted completion time. To estimate the worst-case execution time of a computing activity on a processor, this processor's frequency and the maximum possible number of instructions for this type of activity are used. Concerning communication time predictions, it is necessary to take the maximum availability of links on the path in addition to the input and output memories bandwidths. Then, an estimated worst-case communication time is computed using the maximum data volume for this activity and the lowest available bandwidth on the path. Note that in our test bench, the possibility is left to choose to off-load a request of a busy rack to a less busy neighbour rack, when the additional cost of moving data to and from the receiving rack compensates advantageously for long potential delays in the original rack.

QoS-Aware Load Balancing This strategy is an extension of the previous prediction-based load balancing whose goal was the reduction of application latencies. While the latter allocates the earliest finishing resources to individual requests, this strategy now aims at optimizing a QoS indicator based on the global cost of exceeding deadlines. As mentioned before, the activity requests are of different types with different priorities, and thus a variable degree of flexibility can be left to miss deadlines. This is an alternative to the hard deadline model. This alternative fits better our radar context, which runs activities at variable computing costs and variable requirements on reactivity; as an example, activities related to tracking targets are expected to complete soon enough to avoid loosing the target, while others operate in a human time scale.

Each type of activity has a penalty factor proportional to the extent of the deadline miss, and null if the deadline is met. The goal is now to keep the accumulated penalty as low as possible while executing the same workload on the same computing platform. Then the QoS-based strategy now considers a sequence of n activity requests of potentially different types (and priorities), sorted in trigger time order. We call R_0 the request in the queue with the earliest trigger time and R_n the n^{th} requests in the queue. While the execution order (i.e. trigger time) isn't impacted, the objective is now to determine if the best mapping found should be used for the next request R_0 or a later, more critical one.

- selects the best host processor P_0 for R_0 and estimates its potential deadline miss and resulting penalty Q_{R_0/P_0}
- $\forall 0 < i <= n$ select the best host processor P_i for R_i with P_0 removed from the list of candidate processors and estimates the potential deadline miss of R_i and resulting penalty Q_{R_i/P_i}

- if $Q_{R_i/P_i} + Q_{R_0/P_0} > Q_{R_i/P_0} + Q_{R_0/P_i}$ and if R_i has a larger priority than R_0, processor P_i is allocated to R_0, leaving P_0 free
- else processor P_0 is allocated to R_0

This resource allocation strategy can be seen as an extension of the first load-balancing method as its behaviour will deviate from the non QoS-aware one only when predicting significant penalty overheads.

5 Evaluation

To evaluate the proposed mapping heuristics a simulation framework has been created mostly because the objective is to evaluate the resource management policies themselves, which would be too time consuming if one had to develop and integrate real-time software for each candidate computing platforms. A modular approach has been adopted, allowing us to calibrate the simulation with actual data while having some freedom in the testing done. This section first describes the created simulation environment before discussing some interesting results.

5.1 Simulation Framework

The simulation environment, realized for this study using the Ptolemy II Framework [20], is divided into three main components:

- a request generation module
- the resource management system
- a computing platform simulator

Request Generation A request generation module has been developed to reproduce representative radar scenarios. It emulates the radar management system request submission pattern and generated requests possess properties close to actual ones. As such, it generates dwell requests as if feeding 4 antennas. The type of dwells and the request released dates are randomized to a certain extend, but in a reproducible way. The generator also includes parameters allowing us to represent the complexity of the environment observed by each antenna by manipulating the probability distribution of actual instruction and communication data volumes of dwells treated by this antenna. These volumes are sent directly to the simulator (see *Vol* Fig. 2) and aren't known by the resource manager which has only knowledge of the request type, min/max volume values, trigger time, deadline and priority.

Resource Manager The resource manager represents the core of our work and encompasses both the mapper and predictor module. It contains the implementation of the proposed mapping algorithms and, for comparison purposes, the implementation of a standard static round-robin placement algorithm in addition to the proposed dynamic methods. Moreover, it keeps track of every missed deadline and assigns a penalty to each application execution calculated as follow: $penalty_n = prio_n * (end\_time_n - deadline_n)$ if $end\_time_n > deadline_n$ or 0 otherwise. The applications' priorities being correlated to the operational interest of a dwell, the sum of suffered penalties during a run represents a relevant estimation of a mapping strategy's impact on the QoS provided to the user while running a scenario.

Platform Simulator To both implement the full MAPE loop and evaluate the proposed methods, an execution environment was required. In order to have more control over it and flexibility on testing parameters, a computing platform simulator has been designed. However, to retain meaningful evaluation results and feedback to the predictor, this simulator has been modeled on an actual platform. Moreover, to accurately simulate the activities' execution, their memory access volume as well as interference (i.e. resource access conflicts) with other running activities are represented. Both the platform simulator and the interference model have been validated by domain experts.

To obtain the following results, the considered platform contains 4 racks linked via an Ethernet switch, each one hosting 6 dual-processor boards with one shared memory per board. Inside a rack, all boards communicate also via an Ethernet switch. Note that each rack receives input data from the front-end of one antenna which is stored in an input memory and must send results to a fixed output memory.

5.2 Results Analysis

We now compare the performance of different mapping methods in several operational scenarios as well as the influence of anticipation delay (i.e. the time between the decision taking process and the application deployment) on the quality of results produced. We first identify the four operational scenarios used in the following experiments:

- *SCE 1* where all antennas share an equal, average load.
- *SCE 2* representing a classic shore surveillance scenario with one antenna (pointing towards the coastline) illuminating a high-complexity environment with a very high number of detections, its 2 neighbors harboring average load and the last antenna treating only few detections.
- *SCE 3* where 2 antennas face numerous detections and the other 2 an average amount.
- *SCE 4*, where all antennas have to treat an important quantity of detections.

While *SCE 1* and *SCE 4* are mostly used as reference points, the second scenario *SCE 2* presents a real operational interest as it represents a traditionally problematic case.

Each result presented below comes from the average of 10 simulation runs with different random seed values. During each run 2000 requests are generated, each request containing 1 computing activity and 1 to 2 communication activities, which translates in an average of 5000 activities mapped per run.

Fig.3 shows the average observed latencies of deployed applications while using a standard round-robin static mapping method. As it is clearly visible in *SCE 2* and *SCE 3*, using a pre-determined mapping prevents online resource sharing and can cause a great load unbalance between computing racks. On the other hand, when using the dynamic load-balancing approach described in 4.3.2 (see Fig.4) we can see that resource sharing has been efficiently carried out, narrowing the gap between computing racks load in both *SCE 2* and *SCE 3*. Moreover, we can observe a global latency reduction anywhere between 18% and 32% across all scenarios. This is due to the fact that our mapping method avoids random latency spikes that can happen while using a static mapping method. These spikes can be caused by the execution of a heavy activity on a pre-determined processor which is possibly already busy executing another expensive activity, effectively leading to an increased queuing time. Additionally, the execution time gains provided by the proposed predictive mapping system are only partly due to the resource sharing between racks as, when preventing the inter-rack migration of activities 10 to 20% execution time reduction was still observed.

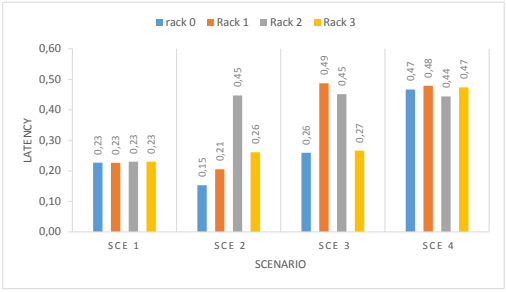

Fig. 3: Latency Using Static mapping

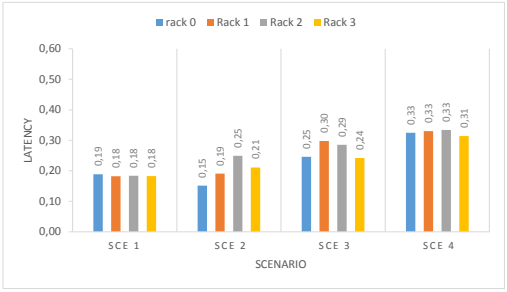

Fig. 4: Latency Using Dynamic Load Balancing

Table 1: QoS results.

		Static Mapping	**Load-Balancing**
SCE 1	penalty	3.21	0
	nb miss	23	0
SCE 2	penalty	69.67	0.17
	nb miss	136.6	3.6
SCE 3	penalty	172.36	2.37
	nb miss	284	28.2
SCE 4	penalty	342.06	31.8
	nb miss	507.4	175

While looking at the QoS provided by both static and predictive methods in Table.1, one can notice an even greater difference in terms of perceived QoS. In the *Static Mapping* column we can observe that, even in a average load scenario (*SCE 1*), a few latency spikes cause some deadline misses, generating some QoS penalties. While the *Load-Balancing* method consistently maintains 0 deadline misses.

In *SCE 2* and *SCE 3* massive differences can be seen in provided QoS with the load balancing method, only missing 3.6 to 28.2 deadlines and the static method between 136.6 and 284 on average. This represent a more than ten times average QoS improvement under the most realistic operational scenarios.

Finally, in *SCE 4* where all racks are overloaded, around one fourth of the deadlines are missed with the static mapping and one eleventh with the dynamic method. Note that these show as well that the use of an efficient load-balancing helps mitigating the effects of an overload. While the load prediction method misses only three times fewer deadlines, it receives only less than 10% the amount of QoS penalty suffered by the static mapping.

Fig.5 shows the impact of computing load of applications on the amount of QoS penalty received for different mapping methods on one computing rack. The computation load is normalized on the computation volume at which the first deadline misses (due to latency spikes) are observed while using the static method. It is possible to observe a steep increase in penalty for all mapping

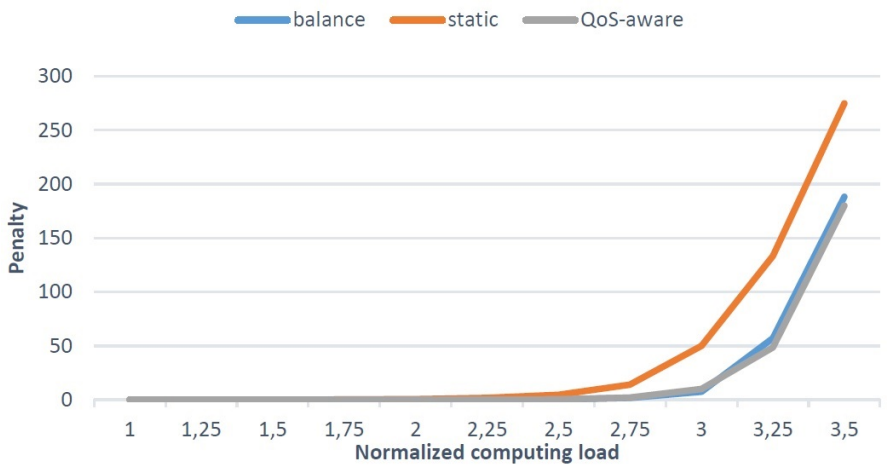

Fig. 5: Impact of Computation Load on the Quality of Result

methods around at around three times this load. At this point we reach the maximum capacity of the computing platform and enter a global overload state. As every application must be executed, numerous computing activities will accumulate inside processors' execution queue and be executed well after their deadlines, generating high penalty.

However, when using a dynamic load-balancing technique, the computing volume at which an overload state is entered differs vastly from the one at which the static method overloads. Even when not taking into account random latency spikes happening for volumes from 1 to 1.5, a constant increase in observed penalty can be noticed for the static mapping from 1.5 onward. On the other hand, when using load-balancing or QoS-aware mapping methods, the first deadline misses and received penalty only appear at the 2.75 mark. This is a straight 83% increase in the computation volume that the platform can absorb before encountering any QoS issue. The QoS-aware mapping method introduced in 4.3.3 shows results close to the first load balancing mapping with an average penalty reduction close to 10%. Moreover, both the QoS-aware and the load-balancing methods tend to mitigate the QoS impact of an overload as they consistently suffer a lower QoS penalty than the static round-robin technique.

As discussed before, the moment at which an application placement decision is taken can greatly influence the quality of load prediction used to take this decision and, thus, impact the quality of produced mapping. This is especially true when operating close to the platform's maximal load. For example, when using a 2.5 volume in *SCE 3*, the average number of missed deadlines can vary from 0 to 38 depending on the anticipation with which the decision is taken. An ideal anticipation of 0 means that the mapping decision is taken instantly at trigger time and yields perfect results when not in an overload state. On the contrary, placing an application as soon as its execution request is received (between 100 to 300 ms before its execution) often yields noticeably inferior results due to more approximate predictions on the platform's future state. The results presented above were obtained using a 10ms anticipation time which provides near optimal results while being a realistic mapping decision delay.

6 Discussion

The results presented in the previous section are encouraging and prove the potential gains of QoS-aware dynamic resource manager for the specialized and demanding systems that are multi-function radars. However, while this work's context seems very specific, the presented predictive load balancing mechanism could be adapted to other domains with similar needs such as high-performance automotive computing platforms. For example, for high performance sensors, this technique could be used to dynamically select the proper algorithms variant that fits best the current observed situation, while maintaining real-time objectives. Moreover, while they operate at completely different scales, cloud environment processing business workflows share some of our use-case constraints [28] and could benefit from an online predictive mapping method.

In addition, this approach opens important opportunities of improvement. First of all, the realized resource management system currently stands in a passive position due to its lack of control over incoming requests and its obligation to execute each of them. Thus, it can only increase the computing load supported by the computing platform before reaching an overload state, with no means to prevent it. Two solutions can be envisaged to address this issue. The first one is the implementation of an admission control unit discarding low priority applications to make room for high priority ones when anticipating an overload. The other envisaged solution is a second control loop between the resource manager and the radar manager. This loop would receive regular updates of the platform state to help determine which type of waveform it could process efficiently as well as to exploit free computing opportunities by launching context-aware useful functions and, thus, enhance the QoS provided to the operator.

While the results of the QoS-aware mapping method are encouraging, one anticipates that its efficiency will largely depend on the relevant weighting of priorities and deadline penalty factors. These parameters being characterized by the radar management system, future work will include the investigation of the influence of these factors when using real-life workloads. Moreover, the computing time prediction algorithms used in the predictor are limited by the worst-case approach used. We are currently exploring a mixed-critical stochastic approach aiming at providing an even better trade-off between QoS provided and predictability of the systems.

To explore these ideas, the simulator presented in this paper is being extended by adding support for heterogeneous processing elements as well as the generation of more complex and diverse workloads. This will allow us to finely tune the prediction model to improve the QoS-aware mapping method presented above as well as to profile the mapping algorithms on the target platform. These profiling data are crucial as they will determine the computing volume limit of the envisaged mapping heuristics. This will directly influence the size and complexity of application graphs that can be considered for further tests. In addition to that, several other QoS metrics, both radar specific (false alarm rate) and generic (processor slack time minimization), are investigated to further improve the mapping results' quality.

Finally, as a centralized resource manager is currently in use, the computing platform's scaling might be an issue. As a fully distributed resources management system is hardly conceivable in the studied use-case, a hierarchical manager composed of both a central unit and locally distributed manager will be considered. Such systems have already been successfully implemented in dynamic large-scale environments such as clouds [26].

7 Conclusion

In this paper we presented a dynamic mapping method for real-time application execution on a heavily-constrained embedded architecture. This method was tested on an AESA radar use-case using a custom simulator. We showed that this approach allows us to obtain lower execution latencies than current mapping solutions while maintaining high predictability and allowing gradual performance degradation in overload scenarios.

8 Acknowledgments

This work was made possible thanks to the support of the Surface Radar Business Line of Thales.

References

1. http://www.uni-siegen.de/dreams/home/
2. http://www.certainty-project.eu/
3. Baruah, S., Li, H., Stougie, L.: Towards the design of certifiable mixed-criticality systems. In: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE. pp. 13–22. IEEE (2010)
4. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed computing **61**(6), 810–837 (2001)
5. Cai, Z., Li, X., Ruiz, R., Li, Q.: A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. Future Generation Computer Systems **71**, 57–72 (2017)
6. Chen, H., Wang, F., Helian, N., Akanmu, G.: User-priority guided min-min scheduling algorithm for load balancing in cloud computing. In: Parallel computing technologies (PARCOMPTECH), 2013 national conference on. pp. 1–8. IEEE (2013)
7. Costache, S., Parlavantzas, N., Morin, C., Kortas, S.: Merkat: A Market-Based SLO-Driven Cloud Platform. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom). vol. 1, pp. 403–410 (Dec 2013). https://doi.org/10.1109/CloudCom.2013.59
8. De Sensi, D., Torquati, M., Danelutto, M.: A reconfiguration algorithm for power-aware parallel applications. ACM Trans. Archit. Code Optim. **13**(4), 43:1–43:25 (Dec 2016). https://doi.org/10.1145/3004054, http://doi.acm.org/10.1145/3004054
9. Gadioli, D., Palermo, G., Silvano, C.: Application autotuning to support runtime adaptivity in multicore architectures. In: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on. pp. 173–180. IEEE (2015)
10. Garca-Valls, M., Cucinotta, T., Lu, C.: Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture **60**(9), 726–740 (Oct 2014). https://doi.org/10.1016/j.sysarc.2014.07.004, http://www.sciencedirect.com/science/article/pii/S1383762114001015
11. Giannopoulou, G., Stoimenov, N., Huang, P., Thiele, L.: Scheduling of mixed-criticality applications on resource-sharing multicore systems. In: 2013 Proceedings of the International Conference on Embedded Software (EMSOFT). pp. 1–15 (Sep 2013). https://doi.org/10.1109/EMSOFT.2013.6658595
12. Gupta, A., Kumar, A., Nagarajan, V., Shen, X.: Stochastic load balancing on unrelated machines. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1274–1285. SIAM (2018)

13. Khemka, B., Friese, R., Pasricha, S., Maciejewski, A.A., Siegel, H.J., Koenig, G.A., Powers, S., Hilton, M., Rambharos, R., Poole, S.: Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system. Sustainable Computing: Informatics and Systems **5**, 14–30 (Mar 2015). https://doi.org/10.1016/j.suscom.2014.08.001, http://www.sciencedirect.com/science/article/pii/S2210537914000420
14. Kousalya, G., Balakrishnan, P., Pethuru Raj, C.: Workflow Scheduling Algorithms and Approaches. In: Automated Workflow Scheduling in Self-Adaptive Clouds: Concepts, Algorithms and Methods, pp. 65–83. Springer International Publishing, Cham (2017), https://doi.org/10.1007/978-3-319-56982-6_4, dOI: 10.1007/978-3-319-56982-6_4
15. Li, H., Baruah, S.: An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems. In: Real-Time Systems Symposium (RTSS), 2010 IEEE 31st. pp. 183–192 (Nov 2010). https://doi.org/10.1109/RTSS.2010.18
16. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. Journal of Grid Computing **13**(4), 457–493 (2015)
17. Lucier, B., Menache, I., Naor, J.S., Yaniv, J.: Efficient online scheduling for deadline-sensitive jobs. In: Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures. pp. 305–314. ACM (2013)
18. Megow, N., Uetz, M., Vredeveld, T.: Models and algorithms for stochastic online scheduling. Mathematics of Operations Research **31**(3), 513–525 (2006)
19. Nasri, M., Brandenburg, B.B.: Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems (outstanding paper). In: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE. pp. 75–86. IEEE (2017)
20. Ptolemaeus, C. (ed.): System Design, Modeling, and Simulation using Ptolemy II. Ptolemy.org (2014), http://ptolemy.org/books/Systems
21. Quan, W., Pimentel, A.D.: A hierarchical run-time adaptive resource allocation framework for large-scale mpsoc systems. Design Automation for Embedded Systems **20**(4), 311–339 (2016)
22. Ren, J., Phan, L.T.X.: Mixed-criticality scheduling on multiprocessors using task grouping. In: Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on. pp. 25–34. IEEE (2015)
23. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. IEEE transactions on cloud computing **2**(2), 222–235 (2014)
24. Skutella, M., Sviridenko, M., Uetz, M.: Unrelated machine scheduling with stochastic processing times. Mathematics of operations research **41**(3), 851–864 (2016)
25. Tang, X., Li, X., Fu, Z.: Budget-constraint stochastic task scheduling on heterogeneous cloud systems. Concurrency and Computation: Practice and Experience **29**(19) (2017)
26. Wang, Z., Su, X.: Dynamically hierarchical resource-allocation algorithm in cloud computing environment. The Journal of Supercomputing **71**(7), 2748–2766 (Jul 2015). https://doi.org/10.1007/s11227-015-1416-x, https://doi.org/10.1007/s11227-015-1416-x
27. Warneke, D., Kao, O.: Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud. IEEE Transactions on Parallel and Distributed Systems **22**(6), 985–997 (Jun 2011). https://doi.org/10.1109/TPDS.2011.65
28. Xu, R., Wang, Y., Huang, W., Yuan, D., Xie, Y., Yang, Y.: Near-optimal dynamic priority scheduling strategy for instance-intensive business workflows in cloud computing. Concurrency and Computation: Practice and Experience **29**(18) (2017)